Feasibility of Flow Decomposition with Subpath Constraints in Linear Time

Daniel Gibney[1]
Georgia Institute of Technology, Atlanta, GA, USA

Sharma V. Thankachan
North Carolina State University, Raleigh, NC, USA

Srinivas Aluru
Georgia Institute of Technology, Atlanta, GA, USA

Abstract

The decomposition of flow-networks is an essential part of many transcriptome assembly algorithms used in Computational Biology. The addition of subpath constraints to this decomposition appeared recently as an effective way to incorporate longer, already known, portions of the transcript. The problem is defined as follows: given a weakly connected directed acyclic flow network $G = (V, E, f)$ and a set $\mathcal{R}$ of subpaths in G, find a flow decomposition so that every subpath in $\mathcal{R}$ is included in some flow in the decomposition [Williams et al., WABI 2021]. The authors of that work presented an exponential time algorithm for determining the feasibility of such a flow decomposition, and more recently presented an $O(|E| + L + |\mathcal{R}|^3)$ time algorithm, where L is the sum of the path lengths in $\mathcal{R}$ [Williams et al., TCBB 2022]. Our work provides an improved, linear $O(|E| + L)$ time algorithm for determining the feasibility of such a flow decomposition. We also introduce two natural optimization variants of the feasibility problem: (i) determining the minimum sized subset of $\mathcal{R}$ that must be removed to make a flow decomposition feasible, and (ii) determining the maximum sized subset of $\mathcal{R}$ that can be maintained while making a flow decomposition feasible. We show that, under the assumption P $\neq$ NP, (i) does not admit a polynomial-time $o(\log |V|)$-approximation algorithm and (ii) does not admit a polynomial-time $O(|V|^{\frac{1}{2}-\varepsilon} + |\mathcal{R}|^{1-\varepsilon})$-approximation algorithm for any constant $\varepsilon > 0$.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Flow networks, flow decomposition, subpath constraints

Digital Object Identifier 10.4230/LIPIcs.WABI.2022.17

Funding This research is supported in part by the U.S. National Science Foundation (NSF) grants CCF-1704552, CCF-1816027 and CCF-2146003.

1 Introduction

The decomposition of flow networks is both a classical problem in Computer Science and an important tool in Computational Biology, where it is used for transcriptome assembly [2, 8, 13, 15, 16, 17, 21]. It was in this last context that the idea of adding subpath constraints to the decomposition of flow networks arose. These constraints allow one to incorporate longer, known portions of a transcript by enforcing that every path in a provided subpath constraint set is a subpath in the flow decomposition (details given in Section 1.2). Williams *et al.* defined the problem of determining whether a flow decomposition under a set of subpath constraints is feasible and, if so, finding a flow decomposition with the minimum number of paths [21, 22]. Minimizing the size of a flow decomposition under subpath constraints is NP-hard due to the same problem being NP-hard without added constraints, even on

[1] Corresponding author

DAGs [19]. For determining the feasibility of a flow decomposition, Williams *et al.* in [21] presented an algorithm running in time exponential in the total overdemand, where the total overdemand is a quantity related to the flows assigned to every edge and the number of subpath constraints. They also posed the question of whether the problem can be solved in polynomial time. The same authors recently answered this question by providing an algorithm that runs in time $O(|E| + L + |\mathcal{R}|^3)$ on a flow network with $|V|$ vertices, $|E|$ edges, and a subpath constraint set $\mathcal{R}$ where the total sum of the subpath lengths is L [22]. A primary contribution of this work is an algorithm that solves the feasibility problem with an improved, linear time complexity that is $O(|E| + L)$.

We will also consider two optimization versions of this problem: minimizing the number of constraints removed while making a flow decomposition feasible and, its dual, maximizing the number of constraints maintained while making a flow decomposition feasible. When applied to transcriptome assembly, the vertices are often used to represent k-mers and the flows assigned to edges correspond to levels of support for the k-mers being adjacent (frequency of the corresponding $k+1$-mers). The paths found through flow decomposition are interpreted as assembled transcripts. As mentioned above, the idea of adding subpath constraints to the flow decomposition comes from trying to incorporate knowledge of longer portions of transcripts [1, 7, 23]. In the case where a flow decomposition under the constraints is not feasible, a natural next step is to eliminate a small number of these constraints in order to make a flow decomposition possible. This would correspond to rejecting some subset of the longer transcripts used for subpath constraints as being either erroneous or not present in the transcripts that caused the given flow network.

Our work studies the inapproximability of these optimization problems. For the problem of removing a minimal number of subpath constraints, we show that a polynomial-time $o(\log |V|)$-approximation algorithm does not exist under the assumption that P $\neq$ NP. We then show that the minimization problem's dual, maximizing the number of subpath constraints maintained, is more difficult to approximate. For the maximization version, there does not exist a polynomial-time approximation algorithm with an approximation factor that is $O(|\mathcal{R}|^{1-\varepsilon} + |V|^{\frac{1}{2}-\varepsilon})$ for any constant $\varepsilon > 0$, assuming P $\neq$ NP.

1.1 Related work

Research on flow decomposition under subpath constraints began with work on finding a minimum path cover (MPC) under subpath constraints. MPC with subpath constraints was initially introduced by Bao et al. as a tool for RNA-sequence assembly [1]. It received further study by Rizzi et al., who considered different weighted cases of the problem and provided polynomial-time algorithms, as well as hardness results for some variations of the problem [14]. Subsequent work has applied these algorithms to assembly [7]. A generalization from path cover to flow decomposition was introduced by Williams et al. [22]. There, the authors observed the deficiency of path decomposition in terms of failing to incorporate frequency information. They propose flow networks and flow decomposition with the minimum number of paths as an improvement. To circumvent this problem's NP-hardness (even without path constraints [19]), the authors provide an FPT algorithm and heuristics as potential solutions.

In our work, we will not be concerned with the problem of finding a minimum number of paths, but rather with the feasibility problem. By avoiding the minimality condition, the problem becomes computationally tractable. Another recent line of related research that avoids this optimization criterion and leads to a polynomial-time solvable problem is finding paths in flow networks that are "safe". These paths are safe in that they appear as subpaths in every flow decomposition. Such safe paths can be enumerated in polynomial time [5, 6, 10].

1.2 Preliminaries

Like in [21, 22], we will only work with directed acyclic graphs (DAGs) in this paper. The reasons for this stem from its application to transcriptome assembly, where the flow networks constructed from a reference genome are DAGs. We define flow networks accordingly. Also, the definition of a flow decomposition here differs from the standard definition in several ways, including that there is no capacity function and that flows are given as part of the flow network itself rather than the decomposition. However, for consistency with [21, 22] we will maintain this terminology.

▶ **Definition 1** (Flow network). *A flow network $G = (V, E, f)$ consists of a (weakly) connected DAG $G = (V, E)$ where V contains special source and sink vertices s and t, and a function $f : E \to \mathbb{N}$ called the flow. The flow satisfies that for all $v \in V \setminus \{s, t\}$, $\sum_{(u,v)\in E} f((u, v)) = \sum_{(v,w)\in E} f((v, w))$. In addition, vertex s has in-degree 0 and vertex t has out-degree 0.*

For an edge $e \in E$, we will refer to $f(e)$ as the flow assigned to e.

▶ **Definition 2** (Flow decomposition). *A flow decomposition $(\mathcal{P}, w)$ of a flow network G is a set of st-paths $\mathcal{P} = \{P_1, ..., P_{|\mathcal{P}|}\}$ and natural numbers $w = \{w_1, \ ..., \ w_{|\mathcal{P}|}\}$, called weights, such that if $e \in E$ is contained exactly in the paths $\{P_{i_1}, ..., P_{i_j}\} \subseteq \mathcal{P}$, then $\sum_{h=1}^{j} w_{i_h} = f(e)$.*

▶ **Definition 3** (Subpath Constraints). *A set of subpath constraints $\mathcal{R} = \{R_1, ..., R_{|\mathcal{R}|}\}$ for a given flow network G is a set of simple paths in G such that for all distinct $R_i, R_j \in \mathcal{R}$, we have R_i is not a subpath of R_j.*

The requirement that no subpath constraint is a subpath of another subpath constraint is a property that will be used in the proof that our linear time algorithm is correct.

▶ **Problem 4** (Flow Decomposition with Subpath Constraints (FDSC)). *An instance $(G, \mathcal{R})$ of FDSC consists of a flow network G and a set of subpath constraints $\mathcal{R}$. The problem is to determine if there exists a flow decomposition $(\mathcal{P}, w)$ of G such that for all subpath constraints $R \in \mathcal{R}$, there exists a path $P \in \mathcal{P}$ where R is a subpath of P.*

If such a flow decomposition exists, we say the instance of FDSC is *feasible*, and we otherwise say it is *infeasible*.

Two optimization variants of this problem are defined below. We consider a solution to the minimization problem to have as the objective value the number of subpath constraints removed from $\mathcal{R}$, and a solution to the maximization problem as the number of subpath constraints maintained.

▶ **Problem 5** (Minimum Subpath Constraint Removal). *Given an instance of FDSC $(G, \mathcal{R})$ and integer $k \geq 0$, determine if there exists a subset of $\mathcal{R}' \subseteq \mathcal{R}$, such that $|\mathcal{R}'| \leq k$ and $(G, \mathcal{R} \setminus \mathcal{R}')$ is feasible.*

▶ **Problem 6** (Maximum Subpath Constraint Retention). *Given an instance of FDSC $(G, \mathcal{R})$ and integer $k \geq 0$, determine if there exists a subset of $\mathcal{R}' \subseteq \mathcal{R}$, such that $|\mathcal{R}'| \geq k$ and $(G, \mathcal{R}')$ is feasible.*

The following definitions will be used in the remainder of this work.

▶ **Definition 7** (Union of Two Paths). *We say two paths $v_{i_1}v_{i_2}...v_{i_k}$ and $v_{j_1}v_{j_2}...v_{j_{k'}}$ can be unioned if there exists suffix-prefix overlap between $v_{i_1}v_{i_2}...v_{i_k}$ and $v_{j_1}v_{j_2}...v_{j_{k'}}$ or between $v_{j_1}v_{j_2}...v_{j_{k'}}$ and $v_{i_1}v_{i_2}...v_{i_k}$ when viewed as strings, i.e., there exists indices i_h and j_g such that either (i) $v_{i_h}v_{i_{h+1}}...v_{i_k} = v_{j_1}v_{j_2}...v_{j_g}$ or (ii) $v_{i_1}v_{i_2}...v_{i_h} = v_{j_g}v_{j_{g+1}}...v_{j_{k'}}$. The union of these two paths in case (i) is $v_{i_1}...v_{i_{h-1}}v_{j_1}...v_{j_{k'}}$, and in case (ii) is $v_{j_1}...v_{j_{g-1}}v_{i_1}...v_{i_k}$. We use $R_i \cup R_j$ to denote the union of subpath constraints R_i and R_j.*

▶ **Definition 8** (Compatible Subpath Constraints). *We say that two subpath constraints R_i and R_j are compatible if R_i and R_j are either vertex disjoint or can be unioned. This is denoted as $R_i \sim R_j$. If two subpath constraints R_i and R_j are not compatible, they are considered incompatible and this is denoted as $R_i \not\sim R_j$.*

Note that by our definition of unioning and compatible subpath constraints, we cannot union two subpath constraints R_i and R_j if they are vertex disjoint, despite them being compatible.

We refer to a set of subpath constraints that can be formed by starting with $\mathcal{R}$ and repeatedly applying zero or more unions as a *unioning of* $\mathcal{R}$. For a particular unioning of $\mathcal{R}$, say $\mathcal{R}'$, and edge $e \in E$, we let $|\mathcal{R}'(e)|$ denote the number of subpath constraints in $\mathcal{R}'$ containing e. We say a subpath constraint set $\mathcal{R}'$ is *maximally unioned* if for all $R_i, R_j \in \mathcal{R}'$, R_i cannot be unioned with R_j. Lastly, we define the length of a path as the number of edges contained in the path.

2 Linear Time Algorithm for FDSC

Our approach will be different from the one taken in [22], although both will result in greedy algorithms. In that work, a graph known as the constraint graph is constructed from the given instance of FDSC. The construction of the constraint graph causes an increased time complexity relative to the algorithm presented here. Also, on the constraint graph, it is possible to derive an equivalent result to Lemma 12, albeit in terms of a path cover on the constraint graph (Lemma 16 in [22]). In the interest of avoiding having to prove that desired equivalence holds outside of the language of the constraint graph, we provide our own proof.

2.1 Equivalence with Finding a Satisfactory Unioning of Constraints

Lemmas 9 - 12 essentially reduce FDSC to the problem of finding an satisfactory unioning of the subpath constraint set.

▶ **Lemma 9** ([21], Corollary 9). *For an FDSC instance $(G, \mathcal{R})$, if for all $e \in E$, $|\mathcal{R}(e)| \leq f(e)$, then $(G, \mathcal{R})$ is feasible.*

▶ **Lemma 10.** *If an instance of FDSC $(G, \mathcal{R})$ is feasible, then there exists a feasible instance of FDSC $(G, \mathcal{R}')$ where $\mathcal{R}'$ is a maximal unioning of $\mathcal{R}$.*

Proof. We can consider the flow decomposition $(\mathcal{P}, w)$ for $(G, \mathcal{R})$ as consisting of (possibly overlapping[2]) st-paths all of weight 1. For duplicate paths we only work with one representative path. For every subpath constraint $R \in \mathcal{R}$, we arbitrarily assign R to one of the representative st-paths $P \in \mathcal{P}$ in the flow decomposition that contains R as a subpath and say that P *satisfies* R.

We first iterate through the representative st-paths in $\mathcal{P}$ in any order. For each representative path, $P \in \mathcal{P}$ we union all non-disjoint subpath constraints satisfied by P. Let the resulting unioning of $\mathcal{R}$ be denoted as $\mathcal{R}'$. We next repeat the following steps until the set of subpath constraints is maximally unioned:

Suppose there exists two compatible, non-disjoint subpath constraints $R_i, R_j \in \mathcal{R}'$ that are not unioned, with the prefix of R_j overlapping with the suffix of R_i. See Figure 1. Let R_i be satisfied by path $P_1 \in \mathcal{P}$ and R_j be satisfied by path $P_2 \in \mathcal{P}$, both having weight 1.

[2] This is permitted under the definition of flow decomposition.

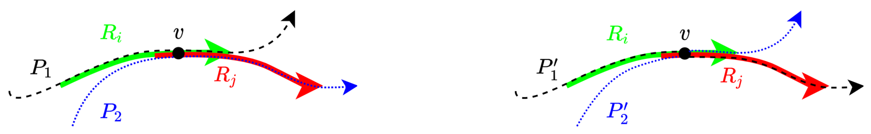


Figure 1 The procedure for swapping portions of paths described in the proof of Lemma 10.

Then there exists some vertex v in common between R_i, R_j, P_1, and P_2. Let $P_1[x, y]$ be the portion of P_1 from vertex x to vertex y (both inclusive) and $P_2[x, y]$ be defined similarly. We next,

1. Remove the paths P_1 and P_2 from $\mathcal{P}$;
2. If not already existing, add the paths $P'_1 = P_1[s, v] \circ P_2[v, t]$ and $P'_2 = P_2[s, v] \circ P_1[v, t]$ to $\mathcal{P}$, where $\circ$ applies concatenation (including v only once);
3. Union R_i and R_j and assign $R_i \cup R_j$ as being satisfied by the representative for P'_1. Additionally, any subpath constraints satisfied by $P_1[s, v]$ or $P_2[v, t]$ are assigned to the representative for P'_1 and any subpath constraints satisfied by $P_2[s, v]$ or $P_1[v, t]$ are assigned to the representative for P'_2 .

For all edges, the sums of path weights are not modified since both P_1 and P_2 had weight 1. Furthermore, other subpath constraints remain satisfied. This is since if a subpath constraint contained v and was being satisfied by either P_1 or P_2, it would have been unioned with R_i or R_j respectively when forming $\mathcal{R}'$. It also still holds that all non-disjoint subpath constraints that are satisfied by the same representative st-path in the decomposition are unioned. Hence, we can repeat Steps 1-3 until the resulting set of subpath constraints is maximally unioned. ◀

▶ **Lemma 11.** *If $\mathcal{R}$ is a maximally unioned subpath constraint set, then $(G, \mathcal{R})$ is infeasible iff there exists an edge $e \in E$ such that $|\mathcal{R}(e)| > f(e)$.*

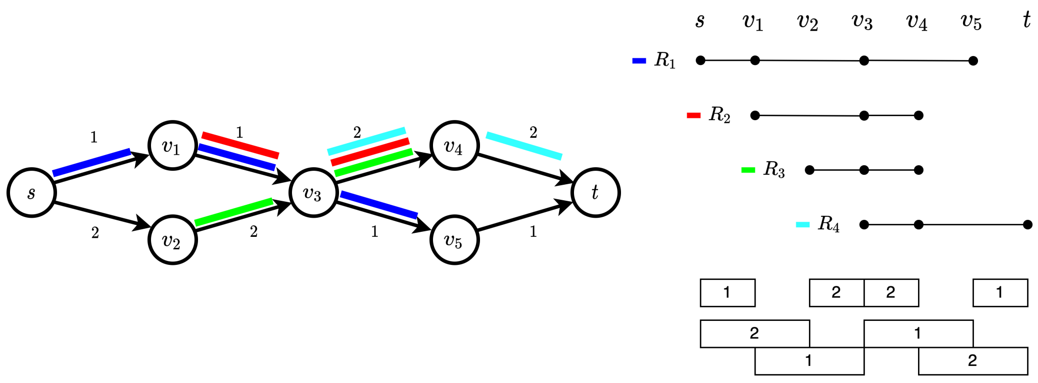
Proof. First consider when $|\mathcal{R}(e)| > f(e)$. Since $\mathcal{R}$ is maximally unioned, every subpath constraint in $\mathcal{R}(e)$ must be satisfied by a distinct path. This implies there must be at least $|\mathcal{R}(e)|$ distinct paths containing e, each with weight at least 1, making the sum of these exceed $f(e)$. Hence, $(G, \mathcal{R})$ is infeasible. In the other direction, if for all $e \in E$, $|\mathcal{R}(e)| \leq f(e)$, then we can directly apply Lemma 9 to obtain that a flow decomposition exists that satisfies the subpath constraints in $\mathcal{R}$. ◀

The main result for this section is Lemma 12.

▶ **Lemma 12.** *For a given FDSC instance $(G, \mathcal{R})$, there exists a flow decomposition satisfying $\mathcal{R}$ iff there exists a unioning $\mathcal{R}'$ of $\mathcal{R}$ where $|\mathcal{R}'(e)| \leq f(e)$ for all $e \in E$.*

Proof. If there exists a flow decomposition satisfying $\mathcal{R}$, then, by Lemma 10, there exists a feasible instance $(G, \mathcal{R}')$ where $\mathcal{R}'$ is maximally unioned. By Lemma 11 we have that $|\mathcal{R}'(e)| \leq f(e)$ for all $e \in E$. In the other direction, if there exists a unioning of $\mathcal{R}$, say $\mathcal{R}'$, such that $|\mathcal{R}'(e)| \leq f(e)$ for all $e \in E$, then, by Lemma 9, there exists a valid flow decomposition for $(G, \mathcal{R}')$. Since the same flow decomposition used for $(G, \mathcal{R}')$ satisfies all of the constraints in $\mathcal{R}$, the instance $(G, \mathcal{R})$ must be feasible as well. ◀

As the first step in determining whether such a unioning exists, we topologically sort the vertices in V. We will henceforth consider the vertices v_1, v_2, ..., $v_{|V|}$ to be in sorted order from smallest to largest. A given constraint $R_i \in \mathcal{R}$ can now be written as $R_i = v_{i_1} v_{i_2} ... v_{i_{|R_i|}}$



■ **Figure 2** (Left) A flow network with four subpath constraints: $R_1 = s\, v_1\, v_3\, v_5$ (dark blue), $R_2 = v_1\, v_3\, v_4$ (red), $R_3 = v_2\, v_3\, v_4$ (green), and $R_4 = v_3\, v_4\, t$ (light blue). Flows for each edge are indicated by the numbers adjacent to the edge. (Right) The same subpath constraints and edge flows, but in the visualization style used in this work.

where these vertices are also considered to be in sorted order from smallest to largest. Using this ordering we can compare the max and min values of a subpath constraint R_i to the max and min values of another subpath constraint R_j.

We can also now more easily visualize an instance of FDSC. An example of this visual representation is shown in Figure 2. This is done by creating for each subpath constraint $R \in \mathcal{R}$ a set of $|R|$ horizontal dots, each positioned horizontally so as to correspond to vertex indices on the number line. For every subpath constraint, we connect each of its dots by a line. In modeling our problem we fix the horizontal position of each set of dots and lines formed from R, but allow their vertical position to be simultaneously shifted. Two non-disjoint constraints can be unioned if we can place one subpath constraint representation on top of the other without a line being placed on top of a dot (or vice versa). At the bottom of this representation we indicate the flow $f((v_\alpha, v_\beta))$ with a box spanning the interval $[\alpha, \beta]$ and containing $f((v_\alpha, v_\beta))$. We can now recast the feasibility problem as trying to find a way to vertically shift these subpath constraint representations on top of one another such that for every interval $[\alpha, \beta]$ the number of lines spanning it is at most $f((v_\alpha, v_\beta))$. This problem representation removes unnecessary information present in the original FDSC instance, such as edges not containing any subpath constraints. In particular, this makes the reduction used in Section 3.1 easier to visualize.

2.2 Our Greedy Algorithm

We first present a high-level overview of the algorithm that forms a subpath constraint set $\mathcal{R}_G$ by unioning subpath constraints in $\mathcal{R}$. As such, we will denote each subpath constraint in $\mathcal{R}_G$ as a subset of the subpath constraints in $\mathcal{R}$, implying that this subset is to be unioned to form a new subpath constraint. This is merely a notational convenience, for example, we denote the unioned subpath constraints $R_{i_1} \cup R_{i_2} \cup \ldots \cup R_{i_k}$ as $\{R_{i_1}, R_{i_2}, \ldots, R_{i_k}\}$. These subsets of $\mathcal{R}$ are disjoint (no subpath constraint appears in two sets) and only contain subpath constraints that are compatible.

Our greedy algorithm *simultaneously minimizes* the number of subpath constraints in $\mathcal{R}_G$ containing any given edge, across all edges. As a result, to determine feasibility, we only need to check for every edge $e \in E$ the number of subpath constraints in $\mathcal{R}_G$ containing e. Implementation details are given in Section 2.4.

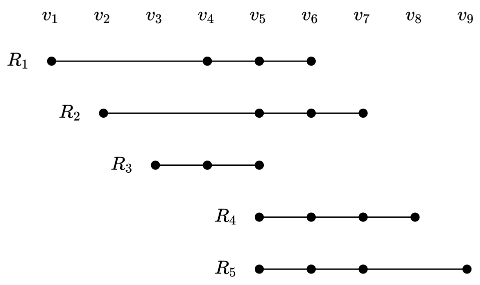


Figure 3 An example sorted set of subpath constraints $\mathcal{R} = \{R_1, R_2, R_3, R_4, R_5\}$ where $R_1 = v_1\, v_4\, v_5\, v_6$, $R_2 = v_2\, v_5\, v_6\, v_7$, $R_3 = v_3\, v_4\, v_5$, $R_4 = v_5\, v_6\, v_7\, v_8$, $R_5 = v_5\, v_6 v_7\, v_9$.

High-Level Algorithm

Recall that all vertices are topologically sorted and indexed according to this order. Sort $\mathcal{R}$ in ascending order according to each subpath constraint's minimum vertex with ties broken arbitrarily. Let R_1, R_2, ..., $R_{|\mathcal{R}|}$ denote the subpath constraints in $\mathcal{R}$ in sorted order from smallest to largest. We next create a new subpath constraint set $\mathcal{R}_G$ that is initially empty. Processing $\mathcal{R}$ in this sorted order from smallest to largest, on iteration i, let

$$\mathcal{R}_{G|\sim i} = \{R \in \mathcal{R}_G : R \sim R_i \text{ and } R \cap R_i \neq \emptyset\}.$$

If $\mathcal{R}_{G|\sim i}$ is non-empty, we update $\mathcal{R}_G$ by unioning R_i with a $R_j \in \mathcal{R}_{G|\sim i}$ that has the largest last vertex, i.e. $j \in \arg\max_h\{\max R_h : R_h \in \mathcal{R}_{G|\sim i}\}$ where $\max R_h$ returns the largest index of any vertex in R_h. If $\mathcal{R}_{G|\sim i}$ is empty, then we add $\{R_i\}$ to $\mathcal{R}_G$.

As an example, consider the subpath constraints $\mathcal{R}$ shown in Figure 3. We create an initially empty set of subpath constraints $\mathcal{R}_G = \emptyset$.

- On iteration $i = 1$, we make $\mathcal{R}_G = \{\{R_1\}\}$.
- On iteration $i = 2$, we make $\mathcal{R}_G = \{\{R_1\}, \{R_2\}\}$.
- On iteration $i = 3$, we make $\mathcal{R}_G = \{\{R_1\}, \{R_2\}, \{R_3\}\}$.
- On iteration $i = 4$, we have to decide whether to union R_4 with $\{R_1\}$, $\{R_2\}$, or $\{R_3\}$. Since $\max\{R_2\} = v_7 > \max\{R_1\} = v_6 > \max\{R_3\} = v_5$, we make $\mathcal{R}_G = \{\{R_1\}, \{R_2, R_4\}, \{R_3\}\}$.
- On iteration $i = 5$, we make $\mathcal{R}_G = \{\{R_1, R_5\}, \{R_2, R_4\}, \{R_3\}\}$.

2.3 Proof of Correctness

Let $\mathrm{OPT}(e)$ denote the minimum number of subpath constraints containing $e \in E$ across all possible ways of unioning $\mathcal{R}$. We claim that after completing all $|\mathcal{R}|$ iterations of the above algorithm, for all edges $e \in E$, we have $|\mathcal{R}_G(e)| = \mathrm{OPT}(e)$. **For the rest of Section 2.3 we fix the edge $\boldsymbol{e = (v_\alpha, v_\beta)}$.** Note that since our greedy algorithm is independent of the choice of e, proving the above claim for e proves it for an arbitrary edge. The proof of the claim involves an exchange argument. For the sake of completeness, we show that an optimal solution can be obtained by iterating over $\mathcal{R}$ in the same order as our greedy algorithm.

▶ **Lemma 13.** *There exists a sequence of unions of subpath constraints in $\mathcal{R}$ that creates a set of subpath constraints $\mathcal{R}'$ such that: (i) $|\mathcal{R}'(e)| = \mathrm{OPT}(e)$, (ii) the process starts with $\mathcal{R}' = \emptyset$ and on the i^{th} iteration, for $i = 1$ to $|\mathcal{R}|$: either unions $R_i \in \mathcal{R}$ with some $R' \in \mathcal{R}'$ created from the previous $i - 1$ iterations, or adds $\{R_i\}$ to $\mathcal{R}'$. Note that this process iterates through $\mathcal{R}$ in the same sorted order as our greedy algorithm.*

Proof. Let $\mathcal{R}''$ be a unioning of $\mathcal{R}$ such that $|\mathcal{R}''(e)| = \text{OPT}(e)$ (not necessarily applying unions in the order described above). Then $\mathcal{R}''$ has a representation of the form $\mathcal{R}'' = \{R''_1, R''_2, ..., R''_k\}$, where each R''_i is the union of some subset of $\mathcal{R}$. We construct a solution as follows: We first create an empty set $\mathcal{R}'$ and process $\mathcal{R}$ in sorted order. When processing R_i, assume that R_i is contained in the subpath constraint $R'' \in \mathcal{R}''$.

- If no previously processed subpath constraint in $\mathcal{R}$ is in R'', we add $\{R_i\}$ to $\mathcal{R}'$.
- If some subset of previously processed subpath constraints in $\mathcal{R}$ are in R'', then we assume inductively that they have already been unioned together, and now union R_i with this partial subset of R''. This is always possible since R_i cannot be vertex disjoint with the partial subset of R''. If it were, then $\min R_i$ would be larger than the maximum vertex in the partial subset of R''. This, together with the sorted order in which $\mathcal{R}$ is being processed, would imply that R'' cannot be a single subpath constraint.

After iterating over all of $\mathcal{R}$, $\mathcal{R}'$ is identical to $\mathcal{R}''$ and the sequence of unions used to obtain it satisfies the stated conditions. ◀

We denote the sequence of unions used in Lemma 13 as the OPT(e)-solution. Let i be the first iteration where our greedy algorithm and the OPT(e)-solution perform a different action on $R_i \in \mathcal{R}$. We will show how to modify the OPT(e)-solution in such a way that it:

- matches our greedy algorithm on the i^{th} iteration;
- does not modify the unions made in earlier iterations;
- has a valid sequence of unions in future iterations;
- does not increase the number of subpath constraints containing e.

Let $\mathcal{R}'$ be the partially constructed set of subpath constraints just prior to the i^{th} iteration resulting from the OPT(e)-solution, and suppose in our greedy algorithm we union R_i with $\{R_{j_1}, ..., R_{j_h}\} \in \mathcal{R}'$ instead of $\{R_{i_1}, ..., R_{i_k}\} \in \mathcal{R}'$ as is done in the OPT(e)-solution. Let the set of subpath constraints unioned with R_i on later iterations in the OPT(e)-solution be denoted as $\{R_{i_{k+1}}, ..., R_{i_t}\}$. Similarly, let the set of subpath constraints in $\mathcal{R}$ unioned with $\{R_{j_1}, ..., R_{j_h}\}$ in later iterations in the OPT(e)-solution be denoted as $\{R_{j_{h+1}}, ..., R_{j_s}\}$. The modification we make to the (completed) OPT(e)-solution is to remove the subpath constraints

$$\{R_{j_1}, ..., R_{j_h}, R_{j_{h+1}}, ..., R_{j_s}\} \text{ and } \{R_{i_1}, ..., R_{i_k}, R_i, R_{i_{k+1}}, ..., R_{i_t}\},$$

then, in the case where $R_{i_k} \cap R_{j_{h+1}} \neq \emptyset$, we add the subpath constraints

$$\{R_{i_1}, ..., R_{i_k}, R_{j_{h+1}}, ..., R_{j_s}\} \text{ and } \{R_{j_1}, ..., R_{j_h}, R_i, R_{i_{k+1}}, ..., R_{i_t}\}.$$

In the case where $R_{i_k} \cap R_{j_{h+1}} = \emptyset$, we instead add

$$\{R_{i_1}, ..., R_{i_k}\}, \{R_{j_{h+1}}, ..., R_{j_s}\}, \text{ and } \{R_{j_1}, ..., R_{j_h}, R_i, R_{i_{k+1}}, ..., R_{i_t}\}.$$

▶ **Lemma 14.** *The modified solution is always valid (no two incompatible subpath constraints are unioned in the modified solution).*

Proof. First, we show that every subpath constraint in the set $\{R_{i_{k+1}}, ..., R_{i_t}\}$ is compatible with every subpath constraint in $\{R_{j_1}, ..., R_{j_h}\}$ (we already know R_i is compatible with $\{R_{j_1}, ..., R_{j_h}\}$ since our greedy algorithm chose it). Suppose for the sake of contradiction that some $R_{j_u} \in \{R_{j_1}, ..., R_{j_h}\}$ is incompatible with some $R_{i_v} \in \{R_{i_{k+1}}, ..., R_{i_t}\}$. See Figure 4 (Left). Because of the sorted order, $\min R_{j_u} \leq \min R_i \leq \min R_{i_v}$. Combining this with

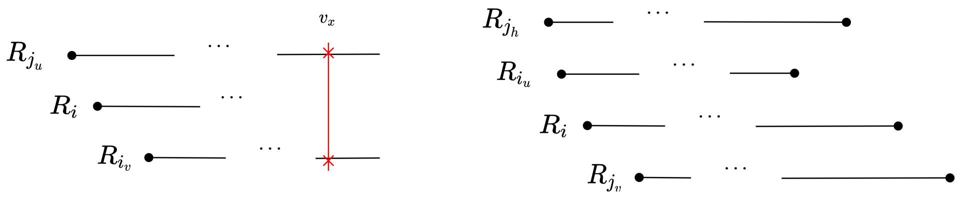


Figure 4 (Left) A visualization of the argument in Lemma 14 that every subpath constraint in $\{R_{i_{k+1}}, ..., R_{i_t}\}$ is compatible with every subpath constraint in $\{R_{j_1}, ..., R_{j_h}\}$. The only relative position that an incompatibility could occur is indicated in red. However, such an incompatibility can not occur since $\max R_i > \max R_{j_u}$. (Right) Every subpath constraint in $\{R_{j_{h+1}}, ..., R_{j_s}\}$ is compatible with every subpath constraint in $\{R_{i_1}, ..., R_{i_k}\}$. Here the relative end positions make an incompatibility impossible as well.

$R_{j_u} \sim R_i$, and $R_i \sim R_{i_v}$, we have that $R_{j_u} \not\sim R_{i_v}$ implies an incompatibility at some vertex v_x where $v_x > \max R_i$ and $v_x < \max R_{j_u}$. However, this implies $\max R_i < \max R_{j_u}$, making $R_i \subset R_{j_u}$, a contradiction.

Next we show that every subpath constraint in the set $\{R_{j_{h+1}}, ..., R_{j_s}\}$ is compatible with every subpath constraint in $\{R_{i_1}, ..., R_{i_k}\}$. Suppose for the sake of contradiction that $R_{i_u} \in \{R_{i_1}, ..., R_{i_k}\}$ is incompatible with $R_{j_v} \in \{R_{j_{h+1}}, ..., R_{j_t}\}$. See Figure 4 (Right). Because our greedy algorithm unions R_i with the subpath constraint having the largest max value, $\max R_{j_h} > \max R_{i_u}$. Also, by the sorted order, we have $\min R_{j_h} < \min R_{j_v}$. Since $R_{j_h} \sim R_{j_v}$, the only way that $R_{i_u} \not\sim R_{j_v}$ is if the incompatibility happens at a vertex greater than $\max R_{j_h}$. However, since $\max R_{j_h} > \max R_{i_u}$, this is not possible. ◀

▶ **Lemma 15.** *The modified solution has at most the same number of subpath constraints containing e as the unmodified* OPT(e)*-solution.*

Proof.

Case 1. $R_{i_k} \cap R_{j_{h+1}} \neq \emptyset$. Recall that the modified solution is

$$\{R_{i_1}, ..., R_{i_k}, R_{j_{h+1}}, ..., R_{j_s}\} \text{ and } \{R_{j_1}, ..., R_{j_h}, R_i, R_{i_{k+1}}, ..., R_{i_t}\}.$$

If both $\{R_{j_1}, ..., R_{j_h}, R_{j_{h+1}}, ..., R_{j_s}\}$ and $\{R_{i_1}, ..., R_{i_k}, R_i, R_{i_{k+1}}, ..., R_{i_t}\}$ contain e in the unmodified OPT(e)-solution, then the number of subpath constraints containing e can not increase in the modified solution. Hence, we only need to consider when the following assumption holds:

▷ Assumption 16. Only one of

$$\{R_{j_1}, ..., R_{j_h}, R_{j_{h+1}}, ..., R_{j_s}\} \text{ and } \{R_{i_1}, ..., R_{i_k}, R_i, R_{i_{k+1}}, ..., R_{i_t}\}$$

contains edge e.

We will show that Assumption 16 implies that only one of $\{R_{i_1}, ..., R_{i_k}\}$, $\{R_i, R_{i_{k+1}}, ...R_{i_t}\}$, $\{R_{j_1}, .., R_{j_h}\}$, $\{R_{j_{h+1}}, ..., R_{j_s}\}$ contains e. This in turn implies that only one of

$$\{R_{i_1}, ..., R_{i_k}, R_{j_{h+1}}, ..., R_{j_s}\} \text{ and } \{R_{j_1}, ..., R_{j_h}, R_i, R_{i_{k+1}}, ..., R_{i_t}\}$$

contains e in the modified solution.

Observe that only a consecutively ordered subset of $\{R_{j_1}, ..., R_{j_h}, R_{j_{h+1}}, ..., R_{j_s}\}$ can contain subpath constraints containing e. To see this, consider R_{j_u}, R_{j_v}, R_{j_w} where $j_u < j_v < j_w$ and R_{j_u} and R_{j_w} contain $e = (v_\alpha, v_\beta)$. We have $\min R_{j_v} \leq \min R_{j_w} \leq$

v_α, and $v_\beta \leq \max R_{j_u} \leq \max R_{j_v}$. These bounds, combined with the compatibility of all three, imply R_{j_v} must contain $e = (v_\alpha, v_\beta)$ as well. The same argument holds for $\{R_{i_1}, ..., R_{i_k}, R_i, R_{i_{k+1}}, ..., R_{i_t}\}$. This observation implies we just need to investigate the two cases below as possible examples where more than one of $\{R_{i_1}, ..., R_{i_k}\}$, $\{R_i, R_{i_{k+1}}, ...R_{i_t}\}$, $\{R_{j_1}, .., R_{j_h}\}$, $\{R_{j_{h+1}}, ..., R_{j_s}\}$ contains e.

(i) Both R_{j_h} and $R_{j_{h+1}}$ contain $e = (v_\alpha, v_\beta)$: We claim that R_i must also contain e. This is since $\min R_i \leq \min R_{j_{h+1}} \leq v_\alpha$, $v_\beta \leq \max R_{j_h} \leq \max R_i$, and $R_{j_h} \sim R_i$. Hence, both $\{R_{j_1}, ..., R_{j_h}, R_{j_{h+1}}, ..., R_{j_s}\}$ and $\{R_{i_1}, ..., R_{i_k}, R_i, R_{i_{k+1}}, ..., R_{i_t}\}$ must contain e, contradicting Assumption 16.

(ii) Both R_{i_k} and R_i contain $e = (v_\alpha, v_\beta)$: We claim that R_{j_h} must contain e. This is since $\min R_{j_h} \leq \min R_i \leq v_\alpha$, $v_\beta \leq \max R_{i_k} \leq \max R_{j_h}$ (where the last inequality is due to our greedy algorithm's selection process), and $R_i \sim R_{j_h}$. Hence, both $\{R_{j_1}, ..., R_{j_h}, R_{j_{h+1}}, ..., R_{j_s}\}$ and $\{R_{i_1}, ..., R_{i_k}, R_i, R_{i_{k+1}}, ..., R_{i_t}\}$ must contain e, contradicting Assumption 16.

This completes the proof for Case 1.

Case 2. $R_{i_k} \cap R_{j_{h+1}} = \emptyset$. Recall that the modified solution is

$$\{R_{i_1}, ..., R_{i_k}\}, \{R_{j_{h+1}}, ..., R_{j_s}\}, \text{ and } \{R_{j_1}, ..., R_{j_h}, R_i, R_{i_{k+1}}, ..., R_{i_t}\}.$$

If e is contained in both $\{R_{j_1}, ..., R_{j_h}, R_{j_{h+1}}, ..., R_{j_s}\}$ and $\{R_{i_1}, ..., R_{i_k}, R_i, R_{i_{k+1}}, ..., R_{i_t}\}$ in the unmodified OPT(e)-solution, then, since $\{R_{i_1}, ..., R_{i_k}\}$ and $\{R_{j_{h+1}}, ..., R_{j_s}\}$ are vertex disjoint, only one of them can contain e. Hence, the number of subpath constraints containing e in the modified solution cannot increase. Again, we only need to consider when Assumption 16 holds and, by the same arguments used when $R_{i_k} \cap R_{j_{h+1}} \neq \emptyset$, we must only look at cases (i) and (ii) from above.

- In (i), R_{i_k} cannot contain e since it is disjoint from $R_{j_{h+1}}$. Combined with the sorted order and $R_{j_{h+1}}$ containing e, this ensures $\{R_{i_1}, ..., R_{i_k}\}$ does not contain e. The previous argument for (i) then applies to $\{R_{j_{h+1}}, ..., R_{j_s}\}$ and $\{R_{j_1}, ..., R_{j_h}, R_i, R_{i_{k+1}}, ..., R_{i_t}\}$ and shows both $\{R_{j_1}, ..., R_{j_h}, R_{j_{h+1}}, ..., R_{j_s}\}$ and $\{R_{i_1}, ..., R_{i_k}, R_i, R_{i_{k+1}}, ..., R_{i_t}\}$ contain e.
- In (ii), $R_{j_{h+1}}$ does not contain e since it is disjoint from R_{i_k}. Combined with the sorted order and R_{i_k} containing e, this ensures $\{R_{j_{h+1}}, ..., R_{j_s}\}$ does not contain e. The previous argument for (ii) then applies to $\{R_{i_1}, ..., R_{i_k}\}$ and $\{R_{j_1}, ..., R_{j_h}, R_i, R_{i_{k+1}}, ..., R_{i_t}\}$ and shows both $\{R_{j_1}, ..., R_{j_h}, R_{j_{h+1}}, ..., R_{j_s}\}$ and $\{R_{i_1}, ..., R_{i_k}, R_i, R_{i_{k+1}}, ..., R_{i_t}\}$ contain e. ◀

Lemmas 14 and 15 imply that the modified sequence of unions remains optimal with respect to the edge e (it becomes a different OPT(e)-solution, but can still be formed by iterating through $\mathcal{R}$ in the same sorted order). Hence, we can repeat the same swapping procedure on the modified solution for the next iteration where a discrepancy occurs with our greedy algorithm.

2.4 Linear Time Implementation via Suffix-Prefix Overlap

We now show how to implement our greedy algorithm via reducing it to suffix prefix overlap, a classical problem in computational biology with an efficient solution [4].

Define the map str that maps an arbitrary subpath constraint $R_i = v_{i_1} v_{i_2} ... v_{i_{|R_i|}}$ onto the string $\text{str}(R_i) = i_1 i_2 ... i_{|R_i|}$. The linear time implementation of our greedy algorithm first performs a linear time sort on $\text{str}(\mathcal{R}) = \{\text{str}(R) : R \in \mathcal{R}\}$ using the subpath constraint's minimum values as keys, with ties broken arbitrarily. Let the sorted strings be denoted $\text{str}(R_1)$,

..., str($R_{|\mathcal{R}|}$). We concatenate a unique symbol $\$_i$ to the end of str(R_i), $1 \leq i \leq |\mathcal{R}|$. Then we concatenate these strings together to form the string $T = \text{str}(R_1)\$_1 ... \$_{|\mathcal{R}|-1}\,\text{str}(R_{|\mathcal{R}|})\$_{|\mathcal{R}|}$ and construct a suffix tree ST over T. Briefly speaking, the suffix tree ST is a compact trie constructed from all suffixes of T. For a given suffix of T, there exists a distinct leaf in ST where the labels assigned to the edges on the root-to-leaf path match the corresponding suffix of T when concatenated. Suffix trees can be constructed in linear time [11, 18, 20], even for strings over integer alphabets [3], like the one used here.

Next, we describe how the suffix tree ST is used to solve the FDSC instance. Let r denote the root of ST. We preprocess ST by marking every node with a branch whose edge has a label that starts with a \$-symbol (including r). We keep pointers from every leaf in ST to its closest marked ancestor. See Figure 5. This preprocessing can be done in linear time via one traversal of ST. We then iterate from $i = 1$ to $|\mathcal{R}|$, and on the i^{th} iteration we start at the leaf ℓ corresponding to the suffix $\text{str}(R_i)\$_i ... \text{str}(R_{|\mathcal{R}|})\$_{|\mathcal{R}|}$ in T. Let ℓ's closest marked ancestor be u.

- If $u = r$, we record that R_i is the start of a new subpath constraint.
- If $u \neq r$ and is marked due to some $\$_h$ we record that R_i should be unioned with R_h. Note that u cannot be marked due to $\$_i$, since this would imply that str(R_i) occurs as a substring twice in T, which would, in turn, imply R_i is completely contained in some other subpath constraint.

This works since for any $j > i$, R_j will either not contain $\min R_i$, or will have an incompatibility before the $\max R_j$ position. This causes a mismatch between the suffix starting at str(R_i) and the suffix starting at str(R_j) prior to the $\$_j$ symbol in the second suffix. Consequently no node marked due to $\$_j$ in ST will be found on the ru-path. Therefore, the lowest $\$_h$ occurring on the path matching R_i in ST indicates R_h where $h < i$, $R_h \sim R_i$, $R_h \cap R_i \neq \emptyset$, and $\max R_h$ is largest.

Once this is completed for all i iterations, we have obtained a sequence of unions that indicates the subpath constraints in $\mathcal{R}_G$. We create a counter for every edge in E. For all $R' = \{R_{i_1}, R_{i_2}, ..., R_{i_k}\} \in \mathcal{R}_G$, we iterate through the set of subpath constraints R_{i_1}, R_{i_2},..., R_{i_k}. For every edge contained in R', we increment the counter for e only the first time it is encountered in R' (actually unioning the subpath constraints in R' is not required). After processing all subpath constraints in $\mathcal{R}_G$, we check for every edge e whether its counter is at most $f(e)$. If this holds for all $e \in E$, we report that the given instance of FDSC is feasible, otherwise, we report that the instance is infeasible.

The initial topological ordering of V requires $O(|V| + |E|)$ time and since G is weakly connected $|E| \geq |V| - 1$. The construction of the suffix tree, followed by the incrementing and checking of counters to determine feasibility, can be done in time proportional to the total length of the subpath constraints in $\mathcal{R}$. Combining these, we obtain Theorem 17.

▶ **Theorem 17.** *An instance $(G = (V, E), \mathcal{R})$ of FDSC can be solved in $O(|E| + L)$ time, where $L = \sum_{R \in \mathcal{R}} |R|$.*

3 Minimizing or Maximizing the Number of Constraints

3.1 Minimizing Subpath Constraint Removal

▶ **Theorem 18.** *There is no polynomial-time approximation algorithm for Minimum Subpath Constraint Removal with an approximation factor that is $o(\log |V|)$ unless $P = NP$.*

Theorem 18 will be proven using a reduction from the Set Cover problem, which is defined as follows: Given a collection $\mathcal{S}$ of subsets S_1, ..., S_m of a universe $U = \{1, 2, ..., n\}$, determine the minimum number of subsets $S_{i_1}, ..., S_{i_k} \in \mathcal{S}$ such that $\cup_{j=1}^{k} S_{i_j} = U$. Well

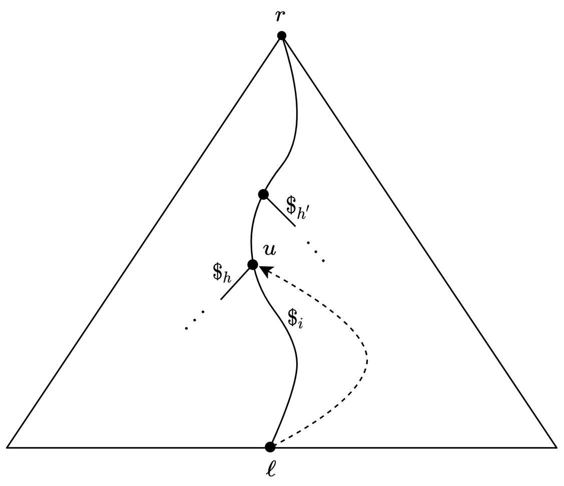


Figure 5 The suffix tree ST for the string $T = \mathrm{str}(R_1)\$_1 ... \$_{|\mathcal{R}|-1}\,\mathrm{str}(R_{|\mathcal{R}|})\$_{|\mathcal{R}|}$ is preprocessed so that every leaf has a pointer to its nearest ancestor that is a marked node. A node is marked if it has a branch whose edge label starts with a \$-symbol.

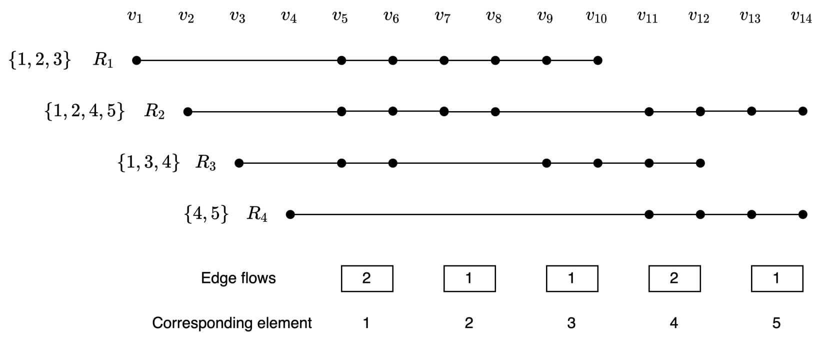


Figure 6 The reduction from the Set Cover instance $\{1,2,3\},\{1,2,4,5\},\{1,3,4\},\{4,5\}$ to an instance of Minimum Subpath Constraint Removal. Only the subpath constraints' relevant corresponding edge flows are shown.

known inapproximability bounds state that no polynomial time $o(\log n)$-approximation algorithm exists assuming P $\neq$ NP. These hold even under the assumption that m and n are polynomially related [9, 12].

Reduction. Let the instance of set cover consist of subsets S_1, ..., $S_m \subseteq U = \{1, 2, ..., n\}$. We construct a flow network G as follows: Let V be initially empty. We first add to V the vertices v_1, v_2,..., v_m, v_{m+1}, ..., v_{m+2n}. For each subpath constraint described below, if an edge specified does not exist in E, we add it to E. Let the set of subpath constraints $\mathcal{R}$ be initially empty. We add to $\mathcal{R}$ the subpath constraints: for $1 \le i \le m$, if $S_i = \{x_1, x_2, ..., x_h\}$,

$$R_i = v_i\ v_{m+2x_1-1}\ v_{m+2x_1}\ v_{m+2x_2-1}\ v_{m+2x_2}\ \cdots\ v_{m+2x_h-1}\ v_{m+2x_h}.$$

Next, we describe the flows assigned to each edge. For the edge $e = (v_{m+2j-1}, v_{m+2j})$, $1 \le j \le n$, we make $f(e) = |\mathcal{R}(e)| - 1$, i.e., the number of subpath constraints containing that edge minus 1. See Figure 6. For the remaining edges created above, we make the flow the number of subpath constraints containing that edge. We now create source and sink vertices s and t. For $v \in V \setminus \{s,t\}$ where $\sum_{(u,v)\in E} f((u,v)) < \sum_{(v,w)\in E} f((v,w))$, we add the edge (s,v) and make

$$f((s,v)) = \sum_{(v,w)\in E} f((v,w)) - \sum_{(u,v)\in E} f((u,v)).$$

For $v \in V \setminus \{s,t\}$ where $\sum_{(u,v)\in E} f((u,v)) > \sum_{(v,w)\in E} f((v,w))$, we add the edge (v,t) and make

$$f((v,t)) = \sum_{(u,v)\in E} f((u,v)) - \sum_{(v,w)\in E} f((v,w)).$$

▶ **Lemma 19.** *There exists a set cover of size k for the instance of Set Cover iff there exist k subpath constraints that when deleted make the resulting modified instance of FDSC feasible.*

Proof. First assume there exist a set cover of size k. For each set S_i in the set cover, delete the subpath constraint R_i. Let $\mathcal{R}'$ denote the modified subpath constraint set. For every $x \in \{1,2,...,n\}$, since x is included in some subset taken for the set cover, the number of subpath constraints containing the edge (v_{m+2x-1}, v_{m+2x}) decreases by at least 1. Hence for all $x \in \{1,2,...,n\}$, $|\mathcal{R}'((v_{m+2x-1}, v_{m+2x}))| \leq |\mathcal{R}((v_{m+2x-1}, v_{m+2x}))| - 1 = f((v_{m+2x-1}, v_{m+2x}))$. Since these were the only edges where the number of subpath constraints containing that edge exceeded the flow, and it only exceeded by 1, we now have that for all $e \in E$, $|\mathcal{R}'(e)| \leq f(e)$. By Lemma 12, this suffices to show that $(G, \mathcal{R}')$ is feasible.

In the other direction, assume there exists $\mathcal{R}' \subseteq \mathcal{R}$, such that $|\mathcal{R}'| \geq |\mathcal{R}| - k$ and $(G,\mathcal{R}')$ is feasible. By Lemma 12, this implies $|\mathcal{R}'(e)| \leq f(e)$ for all $e \in E$. Hence, for all $x \in \{1,2,...,n\}$, $|\mathcal{R}'((v_{m+2x-1}, v_{m+2x}))| \leq f((v_{m+2x-1}, v_{m+2x})) = |\mathcal{R}((v_{m+2x-1}, v_{m+2x}))| - 1$. This implies there exists some subpath constraint that was removed and contained edge (v_{m+2x-1}, v_{m+2x}). Hence, if the removed set of subpath constraints is $|\mathcal{R}| \setminus |\mathcal{R}'| = \{R_{i_1}, ..., R_{i_{k'}}\}$, where $k' \leq k$, we have that $S_{i_1} \cup ... \cup S_{i_{k'}} = \{1,2,...,n\}$. ◀

Assuming the instance of set-cover satisfies the condition that m and n are polynomially related, we have $|V| = m + 2n = n^{\Theta(1)}$. In this case, a polynomial-time algorithm providing a $o(\log |V|)$-approximation for the value k, also provides a $o(\log n)$-approximation for set cover. This completes the proof of Theorem 18.

3.2 Maximizing Subpath Constraint Retention

▶ **Theorem 20.** *For every constant $\varepsilon > 0$, there is no polynomial time $O(|V|^{\frac{1}{2}-\varepsilon} + |\mathcal{R}|^{1-\varepsilon})$-approximation algorithm for Maximum Subpath Constraint Retention, unless $P = NP$.*

Theorem 20 is based on a reduction from the Maximum Independent Set problem defined as follows: Given a graph $G = (V,E)$, determine the maximum sized subset $I \subseteq V$ such that no two vertices in I are adjacent, i.e., I is an independent set. We assume that G is connected, making $|E| \geq |V| - 1$.

Reduction. Let $G = (V,E)$ be a given instance of Maximum Independent Set problem where $V = \{u_1, ..., u_{|V|}\}$ and $E = \{e_1, ..., e_{|E|}\}$. We first create a vertex set V' with vertices v_1, ..., $v_{|V|}$, $v_{|V|+1}$, ..., $v_{|V|+2|E|}$. We next construct an edge set E'. Like in the reduction for Minimum Subpath Constraint Removal, if an edge specified by a subpath constraint does not exist, we add it to E'. Let $\mathcal{R}$ be initially empty. For each $u_i \in V$ we add a constraint to $\mathcal{R}$. Suppose u_i is incident to the edges $e_{i_1}, ..., e_{i_h}$. We make

$$R_i = v_i \; v_{|V|+2i_1-1} \; v_{|V|+2i_1} \; v_{|V|+2i_2-1} \; v_{|V|+2i_2} \; \cdots \; v_{|V|+2i_h-1} \; v_{|V|+2i_h}.$$

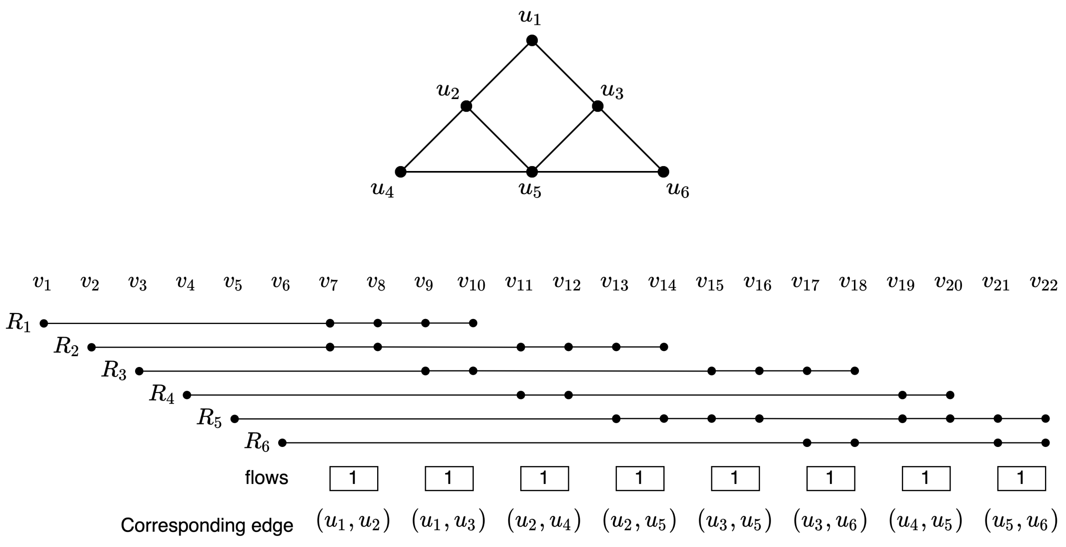

u_1
u_2
u_3
u_4
u_5
u_6
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12} v_{13} v_{14} v_{15} v_{16} v_{17} v_{18} v_{19} v_{20} v_{21} v_{22}
R_1
R_2
R_3
R_4
R_5
R_6
flows
1 1 1 1 1 1 1 1
Corresponding edge
(u_1, u_2) (u_1, u_3) (u_2, u_4) (u_2, u_5) (u_3, u_5) (u_3, u_6) (u_4, u_5) (u_5, u_6)

Figure 7 A reduction from the graph above to an instance of Maximum Subpath Constraint Retention. For every vertex there is a corresponding subpath constraint.

For the edge $e = (v_{|V|+2j-1}, v_{|V|+2j}) \in E'$, $1 \leq j \leq |E|$, we make $|f(e)| = 1$. Just as in the reduction for Minimum Subpath Constraint Removal, we make the flows for the remaining edges the number of subpath constraints containing that edge, then add vertices s and t, edges from s to all vertices in $V' \setminus \{s, t\}$, edges from $V' \setminus \{s, t\}$ to t, and assign flows as needed to create a valid flow network. Let G' be the resulting flow network.

▶ **Lemma 21.** *There exists an independent set of size k for the instance of Independent Set iff there exists k subpath constraints that when maintained (and other subpath constraints deleted) make the resulting modified instance of FDSC feasible.*

Proof. First assume there exist an Independent set I of size k. Maintain the subpath constraints corresponding to the vertices in I and delete the remaining subpath constraints. For edge $(v_{|V|+2h-1}, v_{|V|+2h}) \in E'$ that corresponds to an edge $e_h = (u_i, u_j) \in E$, it can not be that both the subpath constraint corresponding u_i and the subpath constraint corresponding to u_j have been maintained, since that would imply that u_i and u_j are both adjacent and in the independent set I. Hence, maintaining only the subpath constraints corresponding to vertices in I must make the resulting instance of FDSC feasible.

In the other direction, assume there exists $\mathcal{R}' \subseteq \mathcal{R}$, such that $|\mathcal{R}'| = k$ and $(G', \mathcal{R}')$ is feasible. By Lemma 12, all edges $e_h = (u_i, u_j) \in E$ have $|\mathcal{R}'((v_{|V|+2h-1}, v_{|V|+2h}))| \leq f(e) = 1$. Hence either R_i or R_j has been removed. This implies that if we take the subset of vertices in V corresponding to the maintained subpath constraints, every edge in E is incident to at most one vertex in this subset. Hence, this subset is an independent set. ◀

For any constant $\varepsilon > 0$ there does not exist a polynomial-time $O(|V|^{1-\varepsilon})$-approximation algorithm for Maximum Independent Set on a graph $G = (V, E)$ unless P = NP [24]. This combined with the above approximation preserving reduction to an FDSC instance $(G' = (V', E'), \mathcal{R})$ where $\mathcal{R} = |V|$, $|V'| = |V| + 2|E| \leq 4|E| \leq 4|V|^2$, and

$$|\mathcal{R}|^{1-\varepsilon} + |V'|^{\frac{1}{2}-\varepsilon} \leq |V|^{1-\varepsilon} + (4|V|^2)^{\frac{1}{2}-\varepsilon} = |V|^{1-\varepsilon} + 4^{\frac{1}{2}-\varepsilon}|V|^{1-2\varepsilon} = O(|V|^{1-\varepsilon}),$$

proves Theorem 20.

4 Open Problems

A question raised by this work is whether feasibility can be determined in polynomial time on cyclic flow networks. A flow decomposition in this setting may be allowed to contain cycles, so the definition of FDSC should make clear whether constraints can also be cyclic. A polynomial-time algorithm for cyclic graphs would have applications to de novo assembly.

References

1 Ergude Bao, Tao Jiang, and Thomas Girke. BRANCH: boosting RNA-Seq assemblies with partial or related genomic sequences. *Bioinform.*, 29(10):1250–1259, 2013. `doi:10.1093/bioinformatics/btt127`.

2 Elsa Bernard, Laurent Jacob, Julien Mairal, and Jean-Philippe Vert. Efficient RNA isoform identification and quantification from RNA-Seq data with network flows. *Bioinform.*, 30(17):2447–2455, 2014. `doi:10.1093/bioinformatics/btu317`.

3 Martin Farach. Optimal suffix tree construction with large alphabets. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 137–143. IEEE Computer Society, 1997. `doi:10.1109/SFCS.1997.646102`.

4 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997. `doi:10.1017/cbo9780511574931`.

5 Shahbaz Khan, Milla Kortelainen, Manuel Cáceres, Lucia Williams, and Alexandru I. Tomescu. Safety and completeness in flow decompositions for RNA assembly. *CoRR*, abs/2201.10372, 2022. `arXiv:2201.10372`.

6 Shahbaz Khan and Alexandru I. Tomescu. Safety of flow decompositions in dags. *CoRR*, abs/2102.06480, 2021. `arXiv:2102.06480`.

7 Anna Kuosmanen, Ahmed Sobih, Romeo Rizzi, Veli Mäkinen, and Alexandru I. Tomescu. On using longer RNA-Seq reads to improve transcript prediction accuracy. In James P. Gilbert, Haim Azhari, Hesham H. Ali, Carla Quintão, Jan Sliwa, Carolina Ruiz, Ana L. N. Fred, and Hugo Gamboa, editors, *Proceedings of the 9th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2016) - Volume 3: BIOINFORMATICS, Rome, Italy, February 21-23, 2016*, pages 272–277. SciTePress, 2016. `doi:10.5220/0005819702720277`.

8 Wei Li, Jianxing Feng, and Tao Jiang. Isolasso: A LASSO regression approach to RNA-Seq based transcriptome assembly. *J. Comput. Biol.*, 18(11):1693–1707, 2011. `doi:10.1089/cmb.2011.0171`.

9 Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, 1994. `doi:10.1145/185675.306789`.

10 Cong Ma, Hongyu Zheng, and Carl Kingsford. Finding ranges of optimal transcript expression quantification in cases of non-identifiability. *bioRxiv*, 2020.

11 Edward M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM*, 23(2):262–272, 1976. `doi:10.1145/321941.321946`.

12 Jelani Nelson. A note on set cover inapproximability independent of universe size. *Electron. Colloquium Comput. Complex.*, 105, 2007. URL: `https://eccc.weizmann.ac.il/eccc-reports/2007/TR07-105/index.html`.

13 Mihaela Pertea, Geo M Pertea, Corina M Antonescu, Tsung-Cheng Chang, Joshua T Mendell, and Steven L Salzberg. Stringtie enables improved reconstruction of a transcriptome from RNA-Seq reads. *Nature biotechnology*, 33(3):290–295, 2015.

14 Romeo Rizzi, Alexandru I. Tomescu, and Veli Mäkinen. On the complexity of minimum path cover with subpath constraints for multi-assembly. *BMC Bioinform.*, 15(S-9):S5, 2014. `doi:10.1186/1471-2105-15-S9-S5`.

15 Mingfu Shao and Carl Kingsford. Accurate assembly of transcripts through phase-preserving graph decomposition. *Nature biotechnology*, 35(12):1167–1169, 2017.

16 Mingfu Shao and Carl Kingsford. Theory and a heuristic for the minimum path flow decomposition problem. *IEEE ACM Trans. Comput. Biol. Bioinform.*, 16(2):658–670, 2019. doi:10.1109/TCBB.2017.2779509.

17 Alexandru I. Tomescu, Anna Kuosmanen, Romeo Rizzi, and Veli Mäkinen. A novel min-cost flow method for estimating transcript expression with RNA-Seq. *BMC Bioinform.*, 14(S-5):S15, 2013. doi:10.1186/1471-2105-14-S5-S15.

18 Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995. doi:10.1007/BF01206331.

19 Benedicte Vatinlen, Fabrice Chauvet, Philippe Chrétienne, and Philippe Mahey. Simple bounds and greedy algorithms for decomposing a flow into a minimal set of paths. *Eur. J. Oper. Res.*, 185(3):1390–1401, 2008. doi:10.1016/j.ejor.2006.05.043.

20 Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory, Iowa City, Iowa, USA, October 15-17, 1973*, pages 1–11. IEEE Computer Society, 1973. doi:10.1109/SWAT.1973.13.

21 Lucia Williams, Alexandru I. Tomescu, and Brendan Mumey. Flow decomposition with subpath constraints. In Alessandra Carbone and Mohammed El-Kebir, editors, *21st International Workshop on Algorithms in Bioinformatics, WABI 2021, August 2-4, 2021, Virtual Conference*, volume 201 of *LIPIcs*, pages 16:1–16:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.WABI.2021.16.

22 Lucia Williams, Alexandru I. Ioan Tomescu, and Brendan Mumey. Flow decomposition with subpath constraints. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pages 1–1, 2022. doi:10.1109/TCBB.2022.3147697.

23 Ting Yu, Zengchao Mu, Zhaoyuan Fang, Xiaoping Liu, Xin Gao, and Juntao Liu. Transborrow: genome-guided transcriptome assembly by borrowing assemblies from different assemblers. *Genome research*, 30(8):1181–1190, 2020.

24 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory Comput.*, 3(1):103–128, 2007. doi:10.4086/toc.2007.v003a006.